Image Object Classification
CS229 Final Project

Kevin Wilder -- kmwilder@stanford.edu
Adam Krzesinski -- akrzesin@stanford.edu

Goal:

For this project, we attempted to do the ILSVRC2013 classification competition[1], dataset 2, task 2. We are given 1000 categories worth of training images (up to 1300 images per category), and 50000 validation images. For each validation image, we are meant to produce 5 labels, one of which must match one of the image's given labels in order to not be considered an error. As we are short on computing resources, we used a subset of this data -- training on only 250 categories, and validating only on images that included one of these included categories.

Implementation: pre-processing the data:

As our first challenge, we were presented with a set of images of different sizes, hues, lighting conditions, et cetera. The algorithms we were about to use cannot tolerate such variance -- most of them expect a standard normalized feature input.

As a first stab at normalizing our data, we converted each color image to grayscale. Initially grayscale images were unchanged. This would hopefully minimize the effect of lighting conditions (or color vs black-and-white originals) on our downstream classifiers, as well as simplify the processing we'd have to do. For example we'd only have to find centroids on single color values, rather than a three-dimensional RGB matrix.

After this, we felt the next thing to standardize would be the image size, in pixels. We profiled the training and validation image sets, finding an average image size to be ~450x375 pixels, with the smallest images being as little as 64x64, and the highest having over a thousand pixels in each dimension. Eyeballing a few rescaled images, we found that resizing each image to the average image size (rounded to the nearest PATCH_DIM multiple -- discussed below) was a reasonable compromise for standardization.

While we were willing to throw away some data to resize an image, we wanted to preserve the original's aspect ratio -- as a warped object shouldn't be recognizable as the original object. Since not all images fit nicely within our average size, we often had to pad our normalized image in the X or Y dimension to fit our spec. We filled those 'padding' pixels with information repeated from the original image -- the assumption being, pixels near the edge are unlikely to be our target object, but the environment, which would be reasonable to be a repeating pattern.

Finally, now that we've standardized our image size and color, we wanted to prepare it for our downstream processes. We sliced each image into PATCH_DIM * PATCH_DIM pixel patches, and passed a one-dimensional array of these 2D patches on to our next stage.

Implementation: Kmeans:

Inspired by a previously encountered image recognition project ([3]), we intended to run K-means on our collection of patches as a sort of edge detector to find N_CLUSTERS common patterns. We would then transform our images from a vector of patches, to a vector of distances between our patches and our discovered clusters -- so, rather than learning on a featureset of N * PATCH_DIM * PATCH_DIM values, we've reduced to N * N_CLUSTERS real numbers.

Implementation: Principal Component Analysis:

This still leaves us with a large number of features per sample. We investigated using PCA to do an additional layer of feature reduction, but found that applying it directly after K-means clustering didn't improve our results. However, it didn't hurt our validation error by much, and it came with the benefit of reducing the memory footprint of our classifier algorithm, which was hitting limits on our system during some configurations. Hence, we left it in our pipeline. With infinite memory, we would not include it.

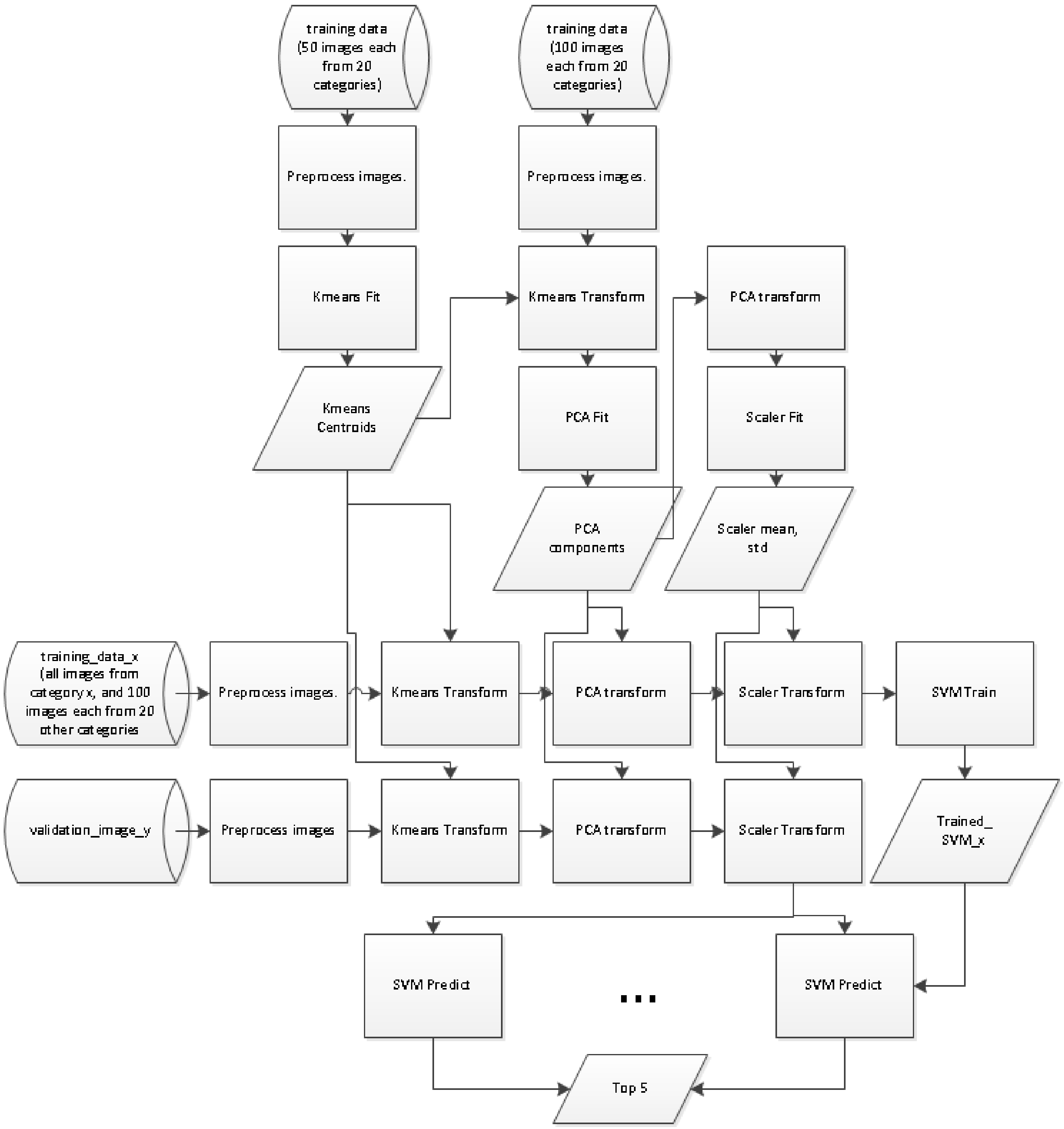
Implementation: Scaler (Standardization):

Finally, to aid in convergence for our SVM, we normalized the training data to have mean 0, variance 1, for each feature. We kept this transformation around, so that validation data would go through an identical transformation.

Implementation: Linear SVM:

We chose to use a linear SVM as our classifier, as they perform very well in high-dimensional feature spaces. Without PCA, our training data proved to be linearly separable.

Implementation: Putting It All Together:



To hit our goal on the computing resources on hand, we chose to do the following: We would run the K-means, PCA, and Scaling algorithms on a subset of our training set, to teach our machine how to see an edge, and how to reduce an image to a standard set of features.

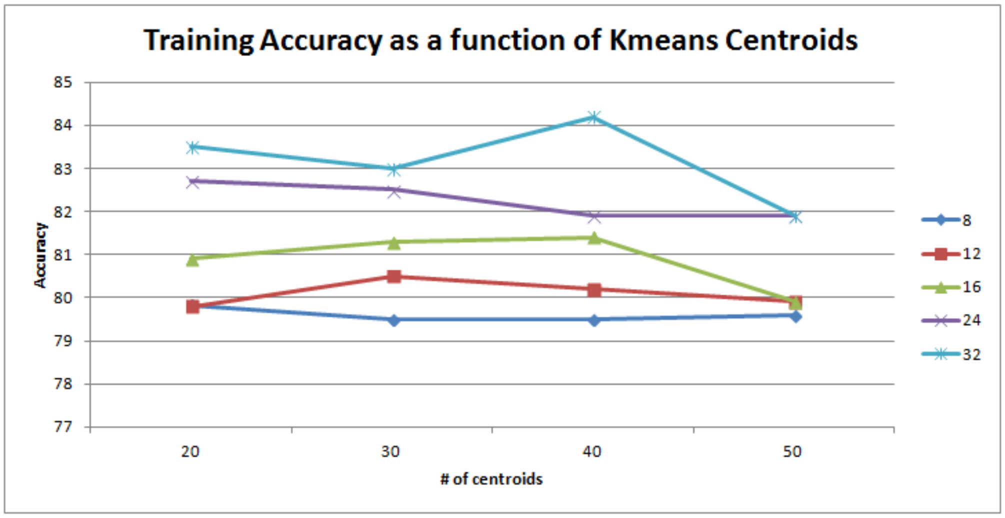
Then, for each category we would want to classify, we'd create a unique SVM, trained to give a 'yes' or 'no' for a given category. The classifier would be trained on a mixture of positive and negative examples (with slightly more negative examples, as more were available.)

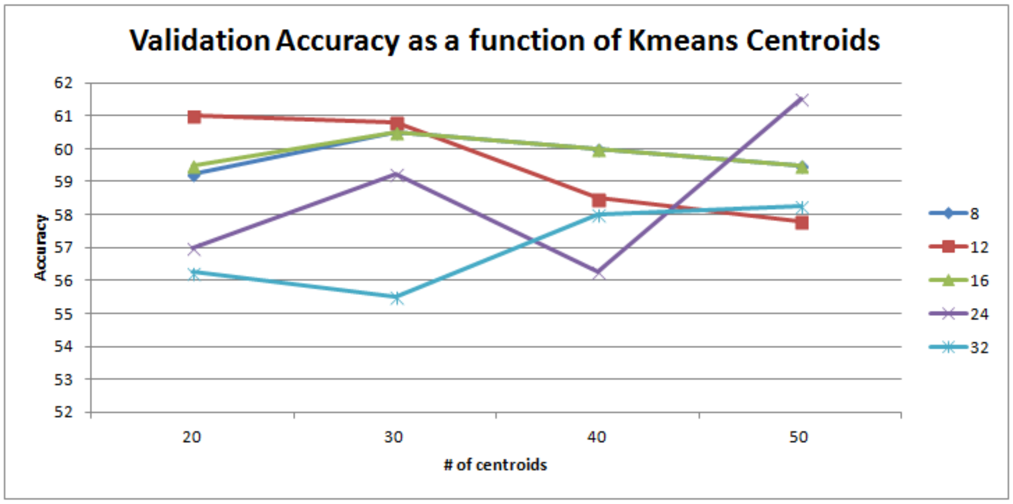
Our goal is, given an image, predict the five most likely categories it could be. We'll do this by pushing the image through the same preprocessing, feature-extraction process as our training data. Then, for each of our classifiers, we'd get not only the prediction, but also the image's distance from the support vector. Our proposal is that the further above the support vector an image is, the more likely it is that the image is what we're classifying.

Tweaking parameters:

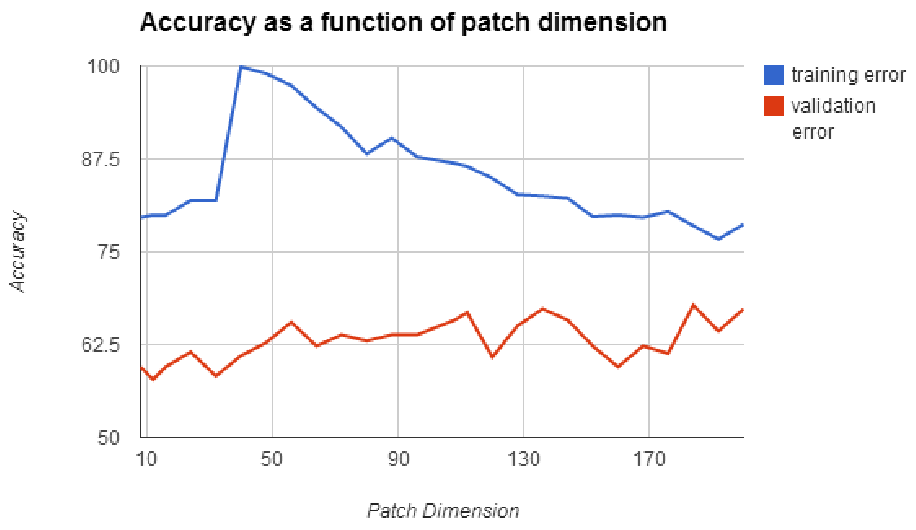
Once we had the initial pipeline functioning, the next step was to adjust the parameters to maximize accuracy. Before doing large-scale testing, our goal was to tweak the parameters to achieve the highest possible single-category validation accuracy. Our pipeline allowed us to adjust the following parameters: Patch dimensions, number of K-means centroids, number of training images, and number of negative training examples.

The following chart shows the training accuracy as a function of the number of centroids for patch dimensions of 8, 12, 16, 24, and 32.



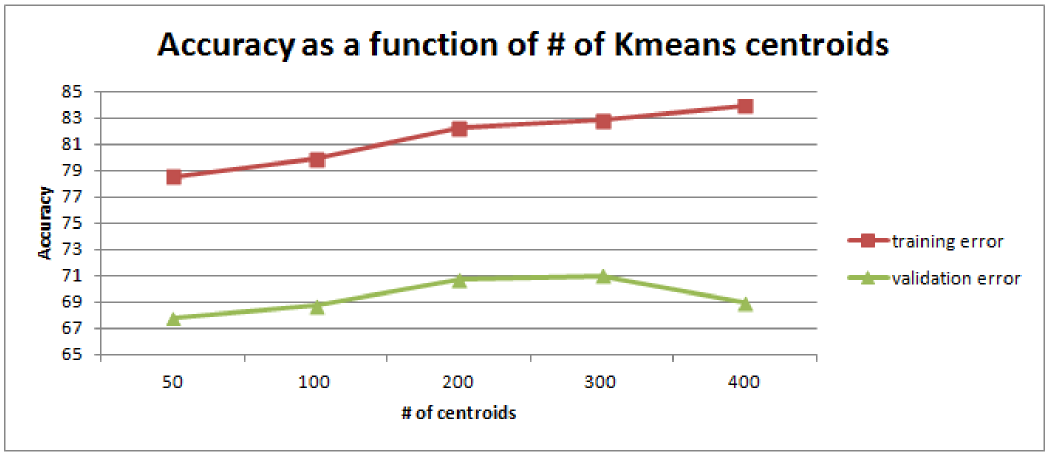


Since there did not seem to be any reasonable relationships here, the next thing we tried was increasing the patch size beyond 32 while keeping the number of centroids constant (50.) The following is a graph of the training and validation accuracy as a function of patch dimension:



As patch dimension increases, we see a drop off in training accuracy while validation accuracy increases slightly. The validation accuracy did seem to increase albeit slowly, but this was promising.

Once we reached an optimal patch size from the above chart, we decided to try adjusting the number of Kmeans centroids. The following chart shows the relationship between the number of centroids and accuracy with an increased patch size (184 pixels):



The accuracy peaked around 300 centroids with a maximum accuracy of 71% and tapered off thereafter.

From this, we believe we found the optimal parameters for our single-category classifier:

Patch dimension: 184 pixels
Number of centroids: 300
Validation accuracy: 71%

Bias and variance:

When initially training our SVM, we were achieving a very high training accuracy. This was initially encouraging, but once we started validating, we noticed that the accuracy was lower than what we were expecting. Based on what we learned in class about bias and variance, this meant we needed to reduce the number of features. To do this, we reduced the number of patches by increasing the size of each patch. The graph above of accuracy vs. patch size confirms this. As patch size increased, the training accuracy decreased, and the validation accuracy slowly increased.

Results:

Using our single-category validation accuracy above to guide us, we chose two configurations to test the full 50000-image dataset, training on our full 1300-image-per-category training set.

With a PATCH_DIM of 184 pixels, using 50 K-means centroids, and without using PCA, our top 5 predictions contained the correct item 816 times, out of 12500 total images, for an accuracy rate of 6.528% (for computational/memory limitation reasons we could not use the optimal single category parameters.)

With a PATCH_DIM of 24 pixels, using 50 K-means centroids, and PCA further reducing the feature set to 200 values, our top 5 predictions contained the correct item 749 times, for an accuracy rate of 5.992%.

Conclusion:

Our final pipeline performs abysmally at image classification. We think a large factor in this is our poor single-category validation accuracy. K-means transformation, followed by a linear classifier, seemed to perform well on small images with few patches. It seems that scaling up to larger pictures requires additional levels of learning, beyond simple edge detection.

Naively taking the distance above our support vectors, and comparing across classifiers, may also be flawed. While we did standardize features across classifiers, with such great classifier inaccuracy, this distance alone loses its meaning.

Citations:

[1] ILSVC2013: http://www.image-net.org/challenges/LSVRC/2013/
[2] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
[3] Piech & Hannun, Visual Cortex, Stanford 2013.
[4] Code for this project can be found at http://stanford.edu/~kmwilder/cs229/